Hardware-Software Implementation of MPEG-4 Video Codec

Seong-Min Kim, Ju-Hyun Park, Seong-Mo Park, Bon-Tae Koo, Kyoung-Seon Shin, Ki-Bum Suh, Ig-Kyun Kim, Nak-Woong Eum, and Kyung-Soo Kim

This paper presents an MPEG-4 video codec, called MoVa, for video coding applications that adopts 3G-324M. We designed MoVa to be optimal by embedding a cost-effective ARM7TDMI core and partitioning it into hardwired blocks and firmware blocks to provide a reasonable tradeoff between computational requirements, power consumption, and programmability. Typical hardwired blocks are motion estimation and motion compensation, discrete cosine transform and quantization, and variable length coding and decoding, while intra refresh, rate control, error resilience, error concealment, etc. are implemented by software. MoVa has a pipeline structure and its operation is performed in four stages at encoding and in three stages at decoding. It meets the requirements of MPEG-4 SP@L2 and can perform either 30 frames/s (fps) of QCIF or SQCIF, or 7.5 fps (in codec mode) to 15 fps (in encode/decode mode) of CIF at a maximum clock rate of 27 MHz for 128 kbps or 144 kbps. MoVa can be applied to many video systems requiring a high bit rate and various video formats, such as videophone, videoconferencing, surveillance, news, and entertainment.

Keywords: MPEG-4, ARM7TDMI, MoVa, 3G-324M.

Manuscript received Oct. 1, 2002; revised Sept. 29, 2003.

This work has been supported by the Korea Ministry of Information and Communication.

Seong-Min Kim (phone: +82 42 860 5347, email: smkim@etri.re.kr), Seong-Mo Park (email: smpark@etri.re.kr), Bon-Tae Koo (email: koobt@etri.re.kr), Kyoung-Seon Shin (email: shinks@etri.re.kr), Ig-Kyun Kim (email: ikkim@etri.re.kr), Nak-Woong Eum (email: nweum@etri.re.kr), and Kyung-Soo Kim (email: kimks@etri.re.kr) are with Basic Research Laboratory, ETRI, Daejeon, Korea.

Ju-Hyun Park (email: pjhyun@mamurian.com) is with Mamurian Design Inc., Seoul, Korea.

Ki-Bum Suh (email: kbsuh@lion.woosong.ac.kr) is with Woosong University, Daejeon, Korea.

I. Introduction

Third generation terminals and mobile networks will be available within the next few years. Increasing attention has been drawn especially to the processing of digital video sequences over the last few years.

As VLSI technologies have advanced, the processing power of general-purpose processors has increased dramatically. Realtime video processing applications tend to be implemented nowadays by software design with the help of a powerful processor. However, this is not a complete solution to fully fulfilling the realtime requirement. Consequently, a powerful instruction set and parallel processing are generally adopted to enhance the computing power for realtime applications.

However, there is a trade-off between hardware and software implementation. Various factors, such as processing speed, flexibility, power consumption, and development cost, should be taken into account. In general, hardware implementation is better than software implementation in power consumption and processing speed [1], [2]. In contrast, software can give a more flexible design solution and also be more suitable for various multimedia applications, such as adding a pre-processing block for handling the input noise components [3], [4].

In order to take full advantage of both hardware and software implementation, we designed the video codec so that each functional module is partitioned in a way appropriate for hardware-software partitioning. The salient feature of codesign is the cooperation of hardware and software modules. In MoVa, the hardware modules are designed to obtain macroblock-based operations. The designed hardware modules work concurrently with ARM7TDMI, an embedded microprocessor core, which performs the software modules. The modules

communicate with the controller via interrupts and parameter passing, and they work together on a simple scheduling and sequencing policy.

MoVa integrates most of the functionality of the simple MPEG-4 profile, ready to support new services with increased performance and reliability and with low power consumption and cost. The ARM7TDMI embedded microprocessor core, the Advanced Microcontroller Bus Architecture (AMBA) [5], and hardwired modules, including the intellectual properties (IPs), are used in the selected solution, because they are useful in coping with tight schedule constraints, in facing competition, and in providing enhanced services.

Section II describes the hardware-software partitioning in MoVa, and section III discusses hardware-software optimization. Section IV covers the pipeline and scheduling, section V the hardware verification, and section VI the implementation. Our results and conclusions are given in sections VII and VIII.

II. Hardware-Software Partitioning

References [6] and [7] described the MPEG-4 video coding flow and functional partition. In the full software implementation of the MPEG-4 video codec, 200 MHz MMX Pentium processors are required for a target bit rate of 128 kbps at 15 frames/s (fps) of quarter common intermediate format (QCIF) sequences [8]. For hardware-software codesign, the codec is first programmed in C and the bottleneck of the processing speed is detoured by embedded hardware modules [9]. Parts of the algorithm are modeled in programming language C for generating a test vector and referring to simulation results after integrating all the hardware modules. Data types are defined as 32-bit, 16-bit, or 8-bit signed or unsigned integers depending on the size hareware elements like registers.

To reduce design time and also increase confidence in the final results, we applied design reuse using hard and soft IP cores. The Hynix ARM7TDMI cell and some generated memories belong to the hard cores. ARM provided the soft cores with register transfer language code, which can be simulated [10], synthesized, and even modified. In addition, we developed AMBA-compliant custom modules for specific video functions.

At first, in order to evaluate the hardware implementation of motion estimation (ME) and motion compensation (MC), we considered the following observations. First, the percentage of the computing power of ME and MC in the MPEG-4 codec is over 80% of the given computing power [11]. Hence, they were better to use in designing hardware to meet the realtime requirement. Second, the discrete cosine transform and quantization (DCTQ) and the inverse quantization and inverse discrete cosine transform (IQIDCT) make up another major complex part with a high computing power requirement. Therefore, we made choice hardware implementation as in most of the dedicated video codec solutions.

We analyzed several hybrid-coding algorithms according to performance requirements. Each function was allocated to suitable hardwired logic or software functions. Functions higher than the macroblock layer were allocated to software functions. Functions lower than the macroblock layer, such as variable length coding (VLC) and variable length decoding (VLD), were implemented in hardwired logic. We then performed algorithm optimization for requirements to consider the tradeoff between the amount of hardware and performance.

In contrast, for the software approach, we designed intra refresh (IR), rate control (RC), error resilience (ER) and error concealment (EC), header VLC (HVLC), and header VLD (HVLD), because they have various forms depending on video compression standards and have an irregular structure not suitable for implementation in hardware. This increased the flexibility of the coding flow. HVLC and HVLD performed the header packing and the header parsing in VLC and VLD, respectively.

1. Hardware Modules

To satisfy both performance and flexibility requirements, we included in the architecture an ARM7TDMI processor core, AMBA, and several hardware modules. AMBA specifies the 32-bit Advanced System Bus (ASB) and the 32-bit Advanced Peripheral Bus (APB). However in MoVa, we used a pseudo-AMBA for more optimization. Because the typical data width of video algorithms is from 16 to 8 bits [11], we modified the ASB to 16 bits and the APB to 8 bits.

Figure 1 shows the MoVa architecture. The codec consists of ARM7TDMI and a wrapper, peripherals for main control, and special purpose modules that support operations required for video picture coding. The special purpose modules perform ME Coarse, ME Fine and MC, DCTQ, VLC, VLD, etc. In addition, there is an on-chip direct memory access controller (DMAC), an external memory interface, peripherals including timers, an interrupt controller, etc. All the modules communicate with each other using the main bus. The VLD buffer memory, the video input/output memory, and the input stream memory interface to the main bus through individual FIFOs to buffer the synchronous dynamic RAM (SDRAM) data traffic. This is strengthened by the fact that loads and stores are performed in parallel with the data computations, involving only small extra times.

The controller ARM7TDMI can be programmed to process various video algorithms, e.g., MPEG-4 and H.263. Instructions

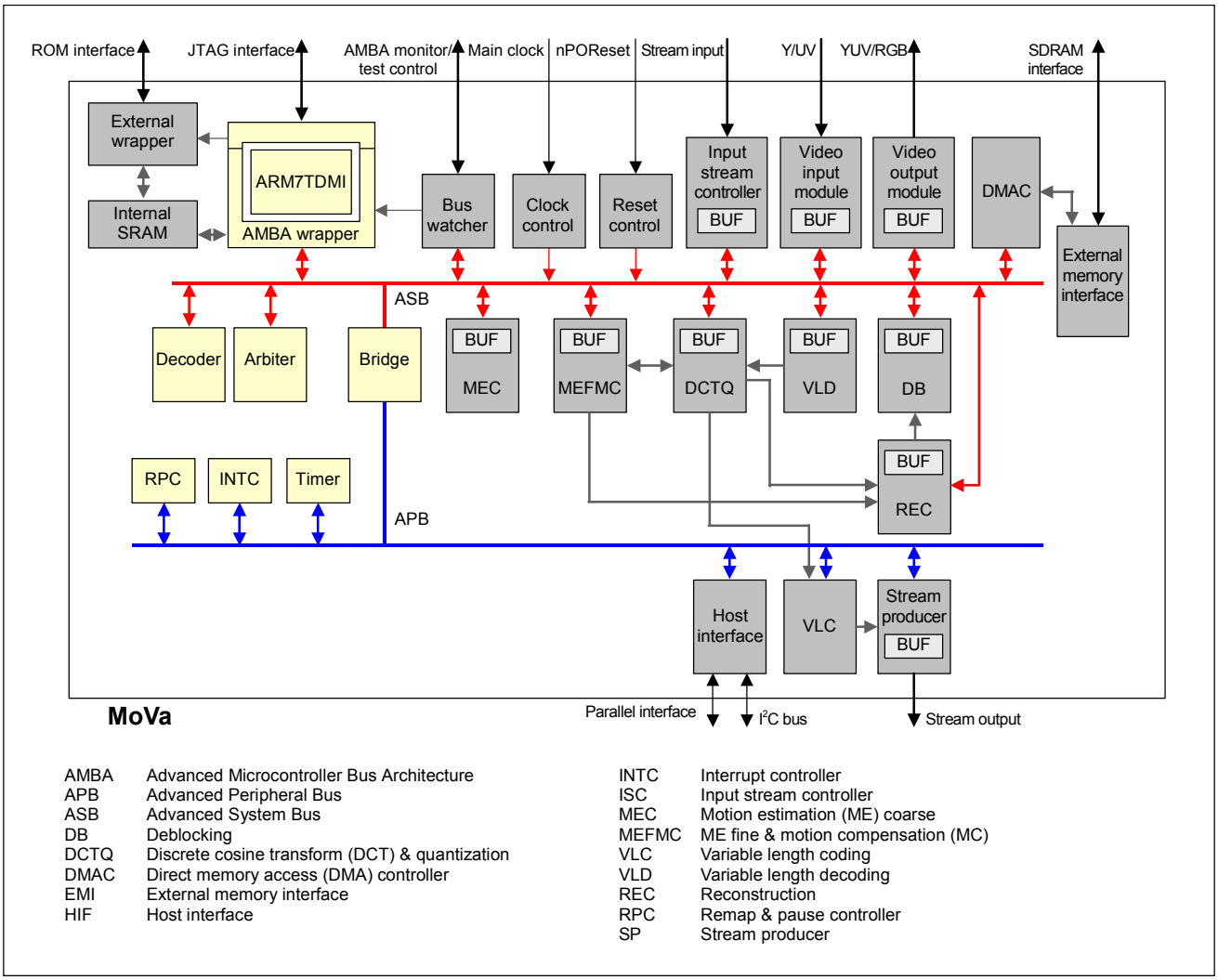

Fig. 1. Video codec architecture.

are executed from an on-chip program memory. An on-chip data memory is used by the controller to interface between hardware and software modules with parameter passing.

The AMBA wrapper, an arbiter, a decoder, and a bridge control the bus master arbitration, module selection signal generation with addressing decoding, and the bridge between two modules. The arbiter determines which masters have access to the bus: an ARM or a DMAC. The decoder runs on a centralized address-decoding module, which generates a select signal for each slave on the ASB bus. The bridge is the only bus master on the APB.

The DMAC arbitrates and schedules all SDRAM accesses concurrent with the controller. The external memory interface provides a 16-bit data bus interface with an SDRAM. An external wrapper provides an 8-bit data bus interface with an ROM whose programs are downloaded into an internal SRAM for program and data memory.

The VLC module reorders DCT coefficients, counts runs of zeros, and performs encoding of the run-value pairs. The VLD module is a decoder that decodes the encoded input bitstream with header parameters received from the controller.

Motion-offset blocks of pixels are fetched from SDRAM in the fast page mode with minimal RAS cycles by the two-dimensional address generator of the DMAC.

For compact performance, we reduce both the number of operations and the number of frame memory accesses. Thus, all hardware modules have minimum-size internal buffers sufficient for parallel operation. The internal buffers also reduce access to the external memory, and therefore reduce power consumption and performance degradation. Hence, we reduce the number of cycles needed for access to below the allowable cycles per macroblock in a 27 MHz operating condition.

A bus watcher provides test and monitoring functions for macroblock signals. In the test mode, AMBA signals, which

can handle each ASB module except the controller, are inputted or outputted through external pins. In the monitor mode, when MoVa is operating normally, internal AMBA signals are monitored through the bus watcher externally. A reset controller indicates the current reset status of MoVa. During reset, the arbiter grants the bus to the default bus master and holds all other grant signals inactive. When MoVa is in a download state, all modules including the controller are inactive.

A clock generator generates the clocks for internal modules to use as follows: 27 MHz, 13.5 MHz, 27 MHz delayed by 1/4 of the period, and 13.5 MHz delayed by 1/4 of the period.

The APB is a secondary bus to the ASB, connected by the bridge. Data access is controlled by a select signal and a strobe signal only. The APB provides an 8-bit data bus interface and includes timers, an interrupt controller, a remap and pause controller, a host interface, a VLC, and a stream producer.

MoVa contains three timers that control the timing of encoding, decoding, and video input. They support maskable interrupt on time-out. The interrupt controller has seven interrupt sources, which are generated by Timer0, Timer1, Timer2, external interrupt, and three soft interrupts. A fast interrupt request [10] is not supported. The remap and pause controller output causes MoVa to enter a "wait for interrupt" state on reset or interrupt, set HIGH on write. For an external host interface, there are bi-directional data transfers between the controller and the external host processor. The external host interface provides a Philips I^2C serial bus interface [12] and an Intel and Motorola parallel interface. Video coding resulting from the VLC is stored in the stream producer (SP) local buffer. The SP outputs bit streams to an off-chip stream buffer.

2. Software Modules

The main controller operates several software modules: a kernel, an initializer, a boot loader, a sequencer, and a scheduler.

The kernel manages how to control its own minimal, standalone, run-time support system for code compiled by an ARM C compiler. It can be assembled using an ARM assembler. In fact, this code depends hardly at all on our target environment and is designed to be easily adapted to any particular ARM-based system. Much of the code is generic to the ARM processor and is completely independent of our ARM-based hardware. This includes i) setting up the initial stack and heap, ii) calling the main function, and iii) program termination. The initializer provides vector and interrupt handlers. The boot loader calls the main function found in the kernel module. The sequencer calls the function of the module interface and executes the function of scheduling and sequence control along with enforcing synchronizing conditions.

The scheduler executes macroblock-based pipeline controls and calls the functions of the module interface for parameter passing. Certain hardware modules consume a fixed time to process input data. These modules are assigned to a fixed delay time based on estimates of their computation times. Examples of these are the DCTQ, the motion estimation coarse (MEC), and the reconstruction (REC). Other modules, such as the VLC, the VLD, the input stream controller (ISC), and the SP, exhibit parameter-dependent behavior.

The main control, the hardware module control, and a few functions, such as header parsing, rate control, error concealment and resilience, motion vector computation, and calculating a frame memory's address for DMAC are implemented by software. The software is simulated with a C/assembly on an ARM toolkit. We identified the time critical modules of the program and hand-coded them into the ARM assembly. We could hand-code the modules programmed by C but we did not hand-code all the modules, using instead some default software coding. We did it this way because the default software had some advantages: it was more readable and made it easier to maintain and document.

ARM7TDMI can make a good microcontroller executing most instructions in a single cycle. Unfortunately, the 32-bit ARM7TDMI microprocessor has a disadvantage: using Thumb code only or using ARM code and Thumb code together causes some implementation problems. We could not implement all functions using Thumb code only. Since the main concern of MPEG-4 target applications is real time, we regard timing optimization more important than code size optimization. There are two phases: the use of C syntax effective for ARM code [13] and scheduling optimization, which has the minimum memory bandwidth. The key to scheduling is to perform hardware and software jobs concurrently. MoVa has two ASB masters, ARM7TDMI and the DMAC. In the AMBA specification, the bus protocol includes pipelined arbitration to ensure that only one master is active on the bus at one time. Every system must have a default bus master, which is granted use of the bus during reset. It is impossible to have more than one master simultaneously operating in AMBA, because it degrades the system performance. For efficient use of an AMBA bus, the wrapper provides an interface between the internal SRAM and the ARM7TDMI, not through AMBA. This makes it possible that two masters may be active simultaneously during an internal mode [5] of the ARM7TDMI master.

The RC is a program-based rate controller for constant bit rate encoding. We present an efficient two-level rate control algorithm that keeps the spatial quality of each frame in a tolerable range. It can be separated into a two-level algorithm, namely the frame-layer and macroblock-layer.

The ER has four major functions: data partition (DP),

reversible VLC (RVLC), resynchronization marker (RM), and intra refresh (IR) [13]. The IR is a method for coping with data loss. This involves sending a fixed number of intra macroblocks in each frame. Annex E of the MPEG-4 standard [13] describes an IR scheme where macroblocks are selected from a motion map. Generation of the motion map is achieved by marking the positions of macroblocks with motion. If the number of macroblocks marked for intra coding exceeds the number of macroblocks encoded, for the next frame, the encoder starts in the same position and begins encoding intra macroblocks, including those marked for intra coding in the previous frame.

The EC resynchronizes the bitstream after the error at the macroblock-level and slice-level and conceals the lost macroblocks. After a macroblock error occurs, the rest of the slice is invoked with default parameters and correctly resynchronized. In case of a packet error, the information of the rest is discarded and correct resynchronization is achieved at the next slice, where all information is reset, so that the next macroblocks will be received correctly.

III. Hardware-Software Optimization

1. Hardware Optimization

The type of ME method and search range significantly affects power consumption and image quality. Therefore, reducing the image pixels from the original image size affects the effective memory bandwidth and the number of computations. In this method the image pixels are subsampled at intervals of two pixels for the horizontal pixels of the input video data, and the subsampling is applied to luminance and chrominance data.

In MoVa, a hierarchical block matching method is used because the full-search block-matching method needs a huge amount of computation. The ME consists of a three-step hierarchical approach, which sequentially operates with search windows of (–14, +14), (–1, +1), and (–0.5, +0.5). The MEC executes the first-step ME which coarsely searches the motion with the subsampled 8×8 current macroblock data in an actual range of +/–14 horizontally or vertically, additionally performs the functions of the MEF skip decision and the inter/intra decision as well as in the unrestricted mode. The MEFMC executes the second- and the third-step ME by the full-search block-matching method for a full MPEG-4 mode, including the advanced prediction mode [14], and searches the fine motion vector up to a half pixel resolution.

The proposed architecture [15] is based on a systolic array. In array architecture, the number of processing elements is dependent on the search range and the calculations per unit time determined by the overall codec performance and power considerations. We implemented the MEC with 8 processing elements and the MEF with 3 processing elements by applying a three-step hierarchical method in MoVa.

MoVa especially incorporates the ME skip scheme, which lowers the power consumption without degrading its performance. The three-step ME operation is skipped only if the predicted sum of the absolute difference (SAD) value of any macroblock currently being processed is not greater than the maximum value of three SAD candidates that correspond to the left, the upper, and the upper-right macroblocks.

When an image is compressed with a low bit rate, there are many not-coded blocks in the video bitstream. In MoVa, since the DCTQ has to be processed prior to the VLC, all blocks must be transformed and quantized even though the DCTQ does not have not-coded blocks. In order to improve the processing speed, the current DCTQ skip is adopted in the inter-frame. In inter-macroblocks, we can predict a not-coded block by comparing the minimum SAD value with a threshold value related to the quantization parameter. Block-based SAD values are obtained from the MEFMC. The smaller the minimum SAD value, the higher the possibility that the inter-macroblock will be designated a not-coded block.

The post filter performs the deblocking operation for only luminance because it has a stronger influence on the luminance boundary than on the chrominance's.

The APB is intended for use with peripherals and modules containing memory-mapped registers only for control without memory dump. APB modules use an 8-bit bus. Using an 8-bit bus instead of a 16-bit bus may result in greater cost and area optimization, but if any modules do not have to dump memory and have few parameter interfaces to request bus bandwidth, the 8-bit bus is sufficient.

2. Software Optimization

There are two categories in software optimization: time and space (memory size). Since the main concern of 3G-324M is for realtime applications, we regard time optimization more important than space optimization.

To increase time and space optimization, we designed freely mixed routines with ARM and Thumb code. At first, we divided all C functions into two uses, time optimization and space optimization, and then compiled them with respective optimization options and compilers. When the functions were optimized for time, they were compiled with an ARM compiler (ARMCC) and time optimization options. However, for space, they were compiled with a Thumb compiler (TCC) and space optimization.

Table 1 shows the size of the code in space optimization for each type of revision we worked on. Most commercial ARM7

series cores have 4 kB or 8 kB embedded memory [16]. In MoVa, the areas of ROM data memory, stack memory, and data memory for module interfaces are about 0.7 kB, 0.3 kB, and 0.5 kB, respectively. Therefore, when another embedded core replaces the ARM7TDMI as a main controller, the program memory of MoVa should be optimized to fit within the size of 6.5 kB.

Table 1. Space optimization.

Compiler / Optimization	ARMCC (bytes)	ARMCC + TCC (bytes)
Time	8,615	-
Space	8,275	-
Time + Space	8,520	7,624
Data	-	7,304
Function	-	6,861
Instruction-level	-	6,837
DMA interface	-	6,854
Final	-	6,467

The original code was too large as it occupied about 125% of the targeted maximum size of 6.5 kB. The details of each optimization type, namely, data, function, instruction-level, and DMA interface optimizations, are presented below.

We optimized space by changing the type of data manipulated by the algorithms. The main objective was to decrease the memory size of the target architecture. In MoVa, variables are based on a 32-bit integer to keep high performance under control. However, the main bus is configured with a 16-bit ASB and an 8-bit APB. If all variables are defined by integer, the most significant 16-bit area is filled with "0x0000." From the point of view of space area, this is not effective. Therefore, if any routine is not critical, it is designed to support computation with 16-bit or 8-bit integers with efficiency. These data types are supported by the ARM compiler through the definition of short and char integer types. Computing with short or char is less efficient than using integer, but the space size is optimized.

For the function optimization type, the pattern of memory dump or parameter passing to each module is similar. Therefore, the functions operating by similar patterns are combined into one function.

The instruction-level optimization is extensively applied in the software for significant time and space optimization. We do not describe these optimizations in detail because they are common knowledge [17]. However, we performed most optimizations manually due to lack of support by the ARM compiler.

A simple example of this transformation is the use of the multiple load and store. In many modules, multiple parameter passings contain memory-mapped registers that can be implemented efficiently with multiple loads and stores. In this case, we force the ARM compiler to use the LDMIA and STMIA instruction by inlining it in assembly.

Since video coding involves massive data, there is a need for global planning and optimization in data storage, traffic, and processing. Data should be defined just with sufficient lengths to save memory space and computation power. Because it is impossible to keep all data in on-chip memory, the most frequently referenced constants, variables, stacks, tables, and working space can be placed in on-chip memory to maximize the effect of fast memory access. Static memory allocation should be adopted for efficiency.

The data transfer method through the ARM bus can be classified into three modes: the internal mode, the sequential mode, and the nonsequential mode. Register-free operation in an internal mode and register access operation in a sequence or a non-sequence mode occurs. A hardware module and a software functional block are operable at the same time by armbreak control in a sequential or non-sequential mode. By using the sequential or non-sequential transfer mode, the space efficiency is lower, but scheduling time is optimized because hardware and software jobs are performed concurrently.

IV. Pipeline and Scheduling

The codec architecture could be realized by accurately calculating memory bus loading and properly scheduling operations of control tasks. For simplifying the design of the timing control of the codec, we adopt a "macroblock-based pipeline" encoding and decoding control method. The pipeline scheduling takes into account solving resource conflicts due to software, hardware, and memory and interface delays.

1. Timing Chart

Figure 2 is a timing chart of the codec for QCIF at 30 fps. MoVa adopts the structure of a fixed time slot for video processing. It has the advantage of simplifying the memory interface because it is necessary not to arbitrate memory access but to control it sequentially. There are two hierarchies, a frame-level and a macroblock-level. At the frame-level, the controller is triggered by a video synchronous interrupt signal to start codec processing with the hardware modules.

The clock frequency in the image sensor's specification [18] for MoVa's application is under 4.5 MHz, which is one sixth

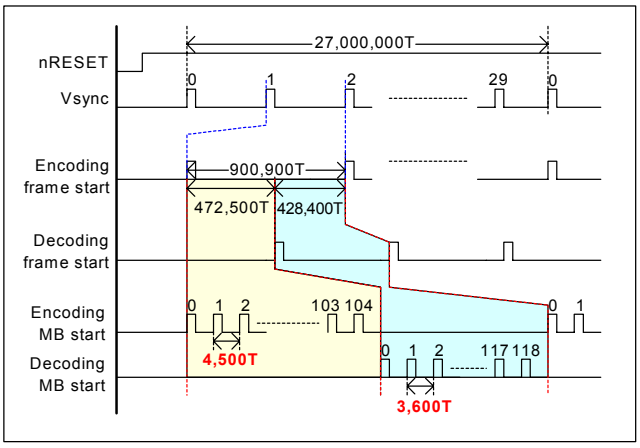

Fig. 2. Timing chart for QCIF at 30 fps.

of MoVa's operating frequency, 27 MHz. In the QCIF format, because the speed of the video encoding is faster than that of the video input and the encoding is impossible to be just operated after the video input of one slice, the encoding for QCIF must be started after downloading the pixel data of more than one frames. The sub-QCIF (SQCIF) is the same as the QCIF in the video data input scheme. The CIF type is similar to QCIF's except for the difference in the number of macroblocks in a frame and the encoding macroblock start time. However, for CIF, the encoding can be done after more than only one slice because the encoding speed is even faster than that of the video input.

2. Pipeline

Figure 3 shows the macroblock-level pipeline flow scheduled by the controller in MoVa. The pipeline stage consists of four stages in encoding and three in decoding. Each stage must be less than 4,500 cycles in encoding and 3,600 cycles in decoding to code 2,970 macroblocks in one second.

The MoVa encoding section sequentially performs the pipeline operation to the MEC, the MEFMC, the MVMVD, the DCTQ, and the VLC, and at every macroblock, the REC and the SP. On the other hand, the decoding section sequentially performs the VLD, the MC, and the IQIDCT, and at every macroblock, the REC and the DB. A feature of this architecture is that internal processing is all executed in parallel. Especially the DCTQ module, VLC module, and VLD module operate at a block (8×8 pixels) base and repeat six times per macroblock. The output data processed at any module is immediately stored at the local memory in the subsequent module. Each local memory is implemented to the fast static RAMs as BUF boxes shown in Fig. 1.

3. Scheduling

A scheduler that reasonably defines a concurrent operation is needed to allocate hardware and software jobs properly for video codec. This scheduler controls the operation of hardware modules by sending parameters such as start, clock gating, and software reset through command registers for flexible control.

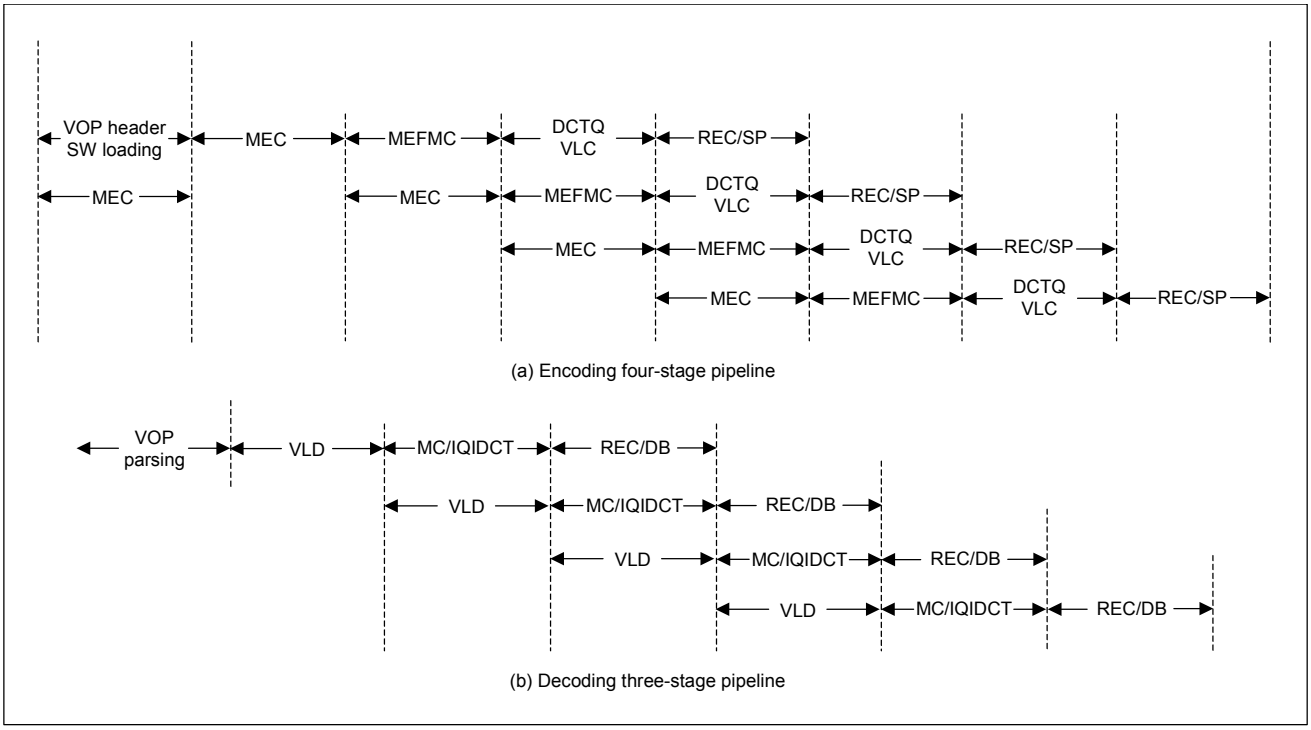

Fig. 3. Macroblock-based pipeline.

For example, if the scheduler writes the 'MEC start' parameter to a command register to start motion estimation, the MEC module continues during the time allocated to one macroblock of MEC video data. Figure 4 shows a macroblock-based scheduling of encoding and decoding.

The height of each block box presents the relative size of memory or bus bandwidth. We used a 16-bit SDRAM data bus width to save memory bandwidth and achieve efficient memory access; thus, all memory dumping for macroblock coding was performed in a macroblock time slot. The scheduling sequentially processes along the vertical axis from "0" value.

When schedule processing is performed, the encoding stage operations and important mark points are as follows:

Stage 1. The current and reference memory data for the MEC is first sent to the MEC buffer memory (MEC-BUF) to perform the coarse motion estimation. After IR for each macroblock has been decided, the MEC module reads from the MEC buffer memory and coarsely searches the motion. In case of the MEC initial latency, the permissible latency set should be adjusted to store the MEC results (MEC-post). The insertion of delays in a pipeline may increase the execution time of an operation; this, in turn will affect dependency. The delay insertion method is applicable only to cases where there is either initial or post latencies.

Stage 2. The integer-pel motion vector and intra flag, the MEC parameters resulting from stage 1, are first stored to data memory (MEC-post). A motion vector obtained from the MEC should be computed before the MEFMC buffer memory for MC of the corresponding macroblock is dumped (the ① symbol in Fig. 4). After the current data and the reference data for the MEFMC is dumped into the MEFMC buffer memory (MEF-BUF) by referencing the motion vector of stage 1, the MEFMC reads the motion vectors, determines the half-pel motion vectors, and performs motion compensation to generate a prediction picture. Video input data, video output data, and input stream data are dumped into each buffer memory (VIM-BUF, VOM-BUF, ISC-BUF). Because these aren't related to any pipeline stage, they can be located at any stage, but the time slot should be fixed.

Stage 3. The half-pel motion vectors and SAD values resulting from the MEFMC at stage 2 are first stored to data memory (MEF-post). Pre-RC determines the target bit rate based on the bits available and the last encoded macroblock bits and computes the quantization parameter (QP) before the DCTQ. The DCTQ operates the DCT, the Q, the IQ, the IDCT, and the AC-DC prediction [7]. In the MVMVD, motion vectors resulting from stage 2 are to be coded differentially. After the DCT and the Q have been completed, the HVLC and the VLC are sequentially operated with the transformed texture data. The parameters of the intra flag, zero motion vector, Dquant, and packet change are used to code the HVLC. Therefore the HVLC should be run after the MEC, the MVMVD, the pre-RC, and the SP (②).

Stage 4. The texture bit number resulting from the VLC is stored for the RC of the next macroblock (VLC-post). After encoding model parameters are updated based on the encoding results of the current macroblock, the post-RC determines whether the next macroblock is skipped according to the current buffer status. If post-RC determines that the next macroblock should be skipped, the reconstructed macroblock is interpolated by copying the pixel data from the macroblock of the same position in a previously reconstructed video object plane (REC-BUF1). If the next macroblock is not skipped, the REC module operates normally (as REC in Fig. 4(b)). The reconstructed macroblock data is stored at the reconstructed SDRAM area (REC-BUF2). Finally, the SP produces a stream from the parameters of the HVLC and the VLC and stores it in the stream output buffer. The post-RC and the SP may not be performed until the result obtained from the VLC is completed for the texture bit (③). A packet change denoting the summation result of bits from the SP in the current macroblock should be produced before the DCTQ, the MVMVD, and the HVLC modules are performed using the parameter (④).

Decoding the pipeline for MoVa is classified into three stages as follows:

Stage 1. The VLD retrieves each VLC stream data stored in the VLD buffer memory (VLD-BUF, VLD). The VLD module cannot be performed until the IQIDCT is completed with data in a buffer between the IQIDCT and the VLD (⑤).

Stage 2. The parsed VLD parameters, such as coded block pattern for luminance (CBPY), coded block pattern for chrominance (CBPC), de-quantization parameter (DP), and not_coded [13], resulting from stage 1 are first stored to data memory (VLD-post). Next, the IQIDCT, the MVMVD, and the MC operate sequentially. The ISC-BUF is the same in stage 2 for encoding. The MC should be completed before the REC is performed at the next pipeline stage, because the REC uses the result data of the MC (⑥).

Stage 3. The VIM_BUF, the VOM_BUF, the REC, and the REC_BUF are the same as in stage 2 and stage 4 for encoding. The DB is a post-filter module and performs along the 8×8 block edges of luminance at the decoder. Each macroblock of luminance is first stored to the DB_BUF to perform the deblocking filtering (DB-BUF1). After the filtering has been

completed (DB), the results are dumped to the SDRAM (DB-BUF2). Finally, the filtered DB data is shifted in the DB buffer memory for the next DB (Post-DB). The REC is performed before the result of the IQIDCT is written on a buffer to inhibit data overflow (⑦).

As mentioned above, the operation queue between modules causes a delay in the total schedule cycles by adding interface cycles. Therefore, a module that receives parameters from only another module passes the parameters by bypassing. Number ⑥ in Fig. 4 exemplifies this case. The input parameters of the MC are passed from the MVMVD output registers by bypassing. This means that bypassing of parameters cannot start until these parameters are available.

V. Verification

After setting up the detailed architecture specification, we completed the design of the video codec using an approach of segmenting the codec design into many relatively small functional modules that we designed and simulated independently. For this reason, having a detailed interface and function specification of the modules before implementation was crucial. Therefore, adopting the AMBA interface was a great advantage.

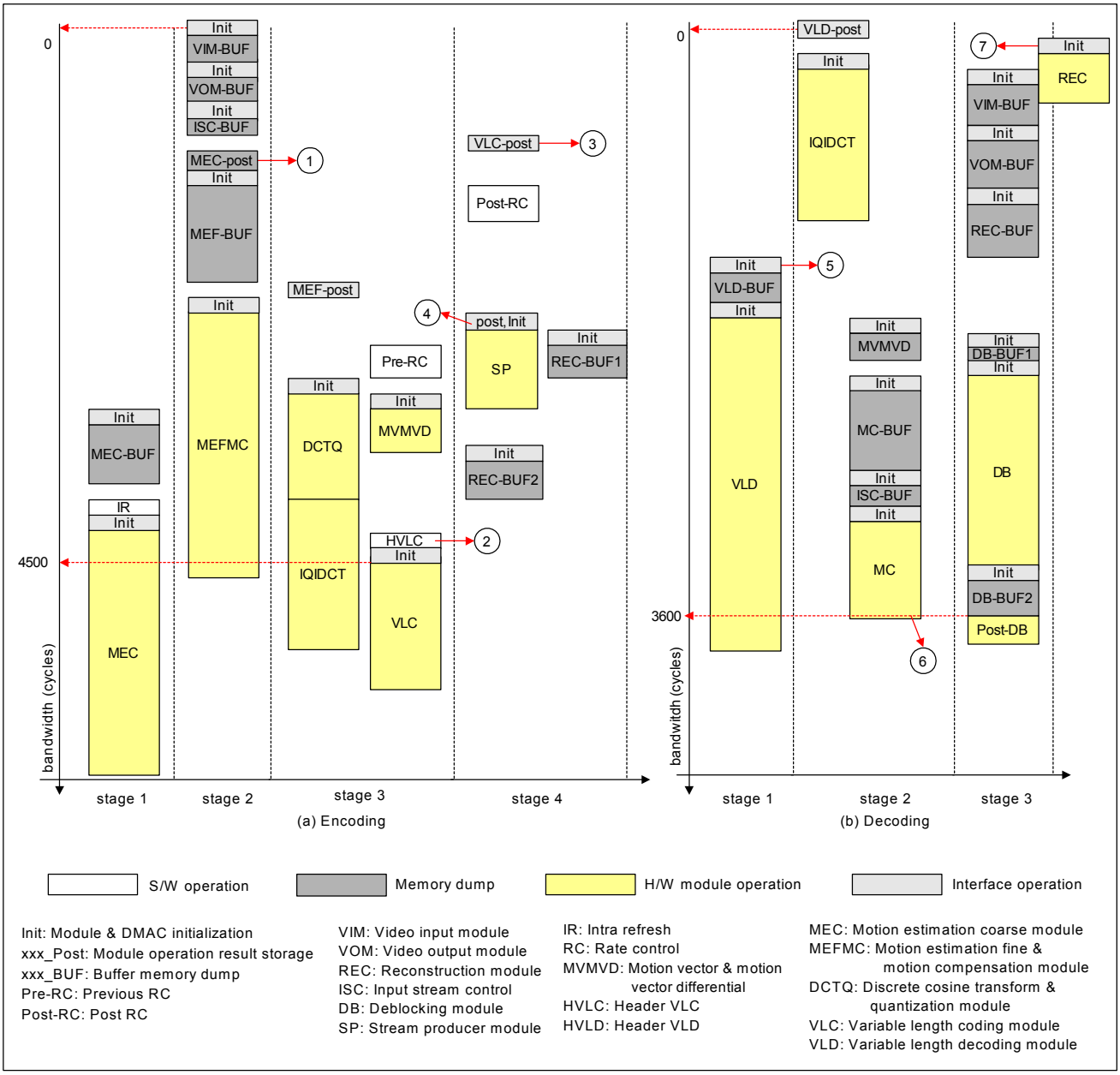

Fig. 4. Macroblock-based pipeline schedule for QCIF.

The hardware design follows the standard application specific integrated circuit (ASIC) design flow based on HDL language and logic synthesis. For the hardware modules, we wrote and simulated the HDL description using the AMBA compliance test bench suites [19]. Each module of the codec was simulated on the function-level. After this initial verification, we connected and tested several modules for AMBA compliance and simulated them again. Seventy-two test vectors were generated by a reference C model and executed in the environment of co-verification. Hardware-software co-verification involved a simulation of the custom hardware described with HDL and the simulation of an ARM7TDMI processor model [20].

For verification, we used test bench models including HDL-models and C-models [21]. The test bench uses an external memory model to simulate the running code on the ARM core in the co-verification environment. In this environment, we compared results from test vectors generated from the reference C model of the hardware architecture with results from the test bench model. Figure 5 illustrates the basic layout of the test bench.

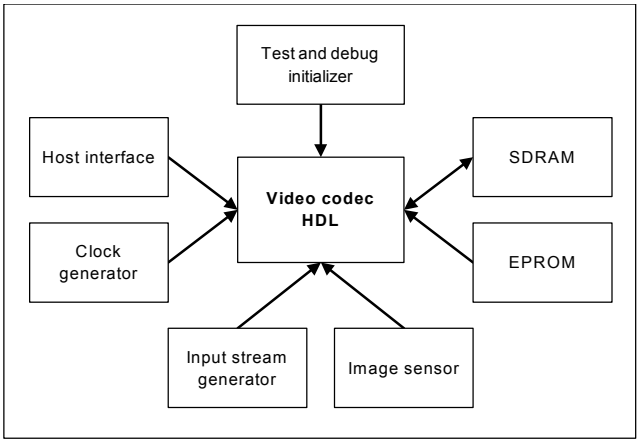

Fig. 5. Block diagram of test bench.

An external memory, 64,000×8 EPROM, is a behavioral Verilog model, and, in general, EPROM loads data from a file specified in the model file. A 16-Mbit SDRAM is a behavioral Verilog model; it loads and stores data from or to two specified bank files. An image sensor [18], an input stream generator, and a host interface [12] are respectively provided as HDL models. Input vectors of the image sensor and the input stream generator are generated from the reference C model of codec. Therefore, after encoding and decoding using the test bench, we compared the results with the results of the reference model. The clock generator makes a video input, a stream input, and a main clock. The interface between ARM signals and external controls for test and debug is established at the initialization of test and debug.

The verification procedure of MoVa becomes increasingly more important and time consuming; therefore, we also concluded that using an FPGA prototyping board would be an efficient verification method. FPGA prototyping offers fast turn-around time and enables several optional choices to be evaluated in a short time and high fault-coverage for hardware and software, which would be impossible using ASIC implementations because of their long turn-around time and high cost.

The full test of this prototyping environment is now in progress. Integrating FPGA prototyping into the design flow of MoVa exploits the respective strengths of both FPGA and ASIC implementations.

VI. Implementation

We implemented the video codec on a hardware-software co-design. The codec can be configured with an arbitrary logic and the programmable modules. On the codec, we placed an ARM7TDMI core, memory modules, and several hardware modules for hardware implementation.

All the hardware modules required a 27 MHz or a 13.5 MHz clock for the encoding and decoding at a frame rate of 30 Hz. The MEFMC, DCTQ, DB, and VLC modules together work at a slow clock speed of 13.5 MHz, so power is saved with these modules.

In APB modules, the processing time of the software interfaced to the hardware becomes longer than the ASB's because an 8-bit bus width is used. This means that the main control consumes a relatively large portion of the total execution time while waiting for the termination of the hardware modules before starting software modules or control codes. In that case, the processing time of the software can be decreased with interfacing between internal memories or hardware modules and ARM7TDMI without AMBA.

Power consumption is a key issue in mobile applications. The codec uses some modes to minimize the overall power consumption.

During the idle modes, MoVa dissipates considerably less power than in the normal operation. The idle mode allows a software application to stop the hardware modules when not in use. The modules have a power manager register field. The register is used to allow software invocation of the power-save mode. During idle mode generated by frame intervals, other on-chip resources are inactive except the video I/O and input stream modules.

The sleep mode offers the greatest power savings to the user. During the sleep mode, MoVa watches for a wake-up event

asserted by an external pin.

The clock-gating scheme typically consumes a large percentage of the codec power. With clock gating, the registers of modules are cut off from the clock at idle times. Eight clocks in the chip were made from three external clocks. Seven of them were gated clocks and were used to control the power-down mode of each functional module according to the operating mode.

The main control and module interface were implemented in C or assembly code to aid in speedy algorithm implementation or modification. A side benefit of this approach is that the user can view the hardware modules and read and write each module register and internal buffer. For example, the software allows the user to start a module through parameter passing and dump the buffer data.

When the memory bandwidth exhaustion is the main bottleneck of the processing speed in the codec, the speed improvement is more closely related to the bus clock speed than the processor's. Therefore, based on the experimental results from the SDRAM interface, we expect that a 54- Mbyte/s SDRAM burst performance can be carried out sufficiently with the total memory bandwidth of the codec.

MoVa has an 8,000-word program/data memory. When the program and data memory are accessed, small memories are better than the large ones for conserving power. As for the memory, the 8,000-word memory is divided into four 2,000-word memories, so these can be accessed with 8-bit, 16-bit, or 32-bit data.

Figure 6 shows the layout pattern of the MPEG-4 codec large scale integrated circuit. Table 2 shows the specifications of the MoVa [21].

Fig. 6. LSI layout pattern.

Table 2. Specifications of MoVa.

Standard	MPEG-4 simple profile @ level 2
Performance	Codec mode: CIF 7.5 fps/QCIF 30 fps Decode mode: CIF 15 fps/QCIF 30 fps
Bit rate	128/144 kbps
Video format	SQCIF/QCIF/CIF
Technology	0.35-μm 4-metal
Gate count	1,700,000 gates
Chip size	110.25 mm^2
Operating frequency	27 MHz
Supply voltage	3.3 V (I/O)
Power consumption	approximately 0.5 W (estimated)
Package	240-pin MQFP

VII. Results

As Table 3 shows, much work needs to be done in developing codecs. The different applications—multimedia, broadcast, acquisition—all have different requirements. However, it must be emphasized that the complete codec was designed, implemented, and tested in less than twelve weeks as opposed to several engineer-years for the commercially available chips. Furthermore, the physical parameters are highly competitive with those of other MPEG designs reported in the literature [22], [24], [25].

In making direct comparisons with the performance and functionality of commercially available MPEG-4 solutions, the reader is reminded that overall focus of the research presented here has been the development of the clock rate and the performance rather than the power consumption. From a technology perspective, MoVa consumes less power than the current standard. The performance of MoVa is directly attributable to the fact that the system clock rate was constrained to 27 MHz.

Codecs like MoVa and emblaze have the DB module; by experimenting with the post-filter, we found it to improve the video quality. This was approximately 3 dB better than similar codecs without the deblocking filter.

All codecs but MoVa have a maximum performance of QCIF 15 fps, but typically it is not easy to support an application having a cinema-level video quality. Unlike all other reviewed codecs, MoVa supports SQCIF and CIF video formats and an AMBA bus interface. This improves the ability to extend application areas and easily upgrade performance because of the flexible and easy bus interface.

Table 3. Performance comparison.

Author \ Item	Size (mm^2)	Process technology	Gate count including memories (million transistors)	Clock rate (MHz)	Power consumption (mW)	Performance (fps for QCIF)
MoVa	10.5 ×10.5	0.35 μm	6.80	27	500	30
Toshiba [22][23]	10.84 ×10.84	0.25 μm	20.50	60	240	15
Matsushita [24]	8.7 ×8.7	0.18 μm	-	54	50	15
Emblaze [25]	-	0.18 μm	-	-	200	15

* Toshiba's chip has on-chip embedded DRAM; thus the chip size and the power consumption are over-estimated.

VIII. Conclusion

We implemented MoVa using a cell-based method with a 0.35-μm CMOS process technology. MoVa contains about 1.7 million gates and is housed in a 240-pin MQFP package. It complies with the MPEG-4 SP@L2 standard. It has a comparatively low power consumption that is achieved by using a low operation frequency. The main contribution of this research is to allow the construction of an MPEG-4 codec that can represent reasonable economy for the mobile phone and the advanced personal digital assistant. Our future research will focus on advancing the performance and reducing the complexity of MoVa. Moreover, by employing embedded DRAM, we will further reduce power consumption.

References

[1] Yil Suk Yang et al., "A Serial Input/Output Circuit with 8 bit and 16 bit Selection Modes," *ETRI J.*, vol. 24. no. 6, Dec. 2002, pp. 462-464.

[2] Jongdae Kim et al., "A Novel Process for Fabricating a High Density Trench MOSFETs for DC-DC Converters," *ETRI J.*, vol. 24. no. 5, Oct. 2002, pp. 333-340.

[3] J. H. Jeng et al., "Multimedia ASIP SoC Codesign Based on Multicriteria Optimization," *ICCE 2001*, June 2001, pp. 342-343.

[4] Kyung Jin Byun et al., "Noise Whitening-Based Pitch Detection for Speech Highly Corrupted by Colored Noise," *ETRI J.*, vol. 25. no. 1, Feb. 2003, pp. 49-51.

[5] ARM, *Advanced Microcontroller Bus Architecture (AMBA) Specification*, Doc. Num.: ARM IHI 0001D, 1997.

[6] T. Sjkora, "The MPEG-4 video standard verification model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, Feb. 1997, pp. 19-31.

[7] MPEG Video Group, *MPEG-4 Video Verification Model*, MPEG96/N1380, Oct. 1996.

[8] Tatsuji Moriyoshi et al., "Real-Time Software Video Codec with a Fast Adaptive Motion Vector Search," *IEEE Workshop Signal Processing Syst., SIPS 99*, 1999, pp. 44-53.

[9] K. Ramkishor and V. Gunashree, "Real Time Implementation of MPEG-4 Video Decoder on ARM7TDMI," *Proc. of 2001 Int'l Symp. Intelligent Multimedia, Video and Speech Processing*, May 2001, pp. 522-526.

[10] ARM, *Example AMBA System Technical Reference Manual*, Micropack v.1.3, Doc. Num.: ARM DDI 0138A, 1998.

[11] J. Kneip et al., "The MPEG-4 Video Coding Standard–a VLSI Point of View," *IEEE Workshop Signal Processing Syst., SIPS 98*, 1998, pp. 43-52.

[12] Philips, PCF8584 I2C-bus Controller Data Sheet, Oct. 1997.

[13] R. Talluri, "Error-Resilient Video Coding in the ISO MPEG-4 Standard," *IEEE Commun. Mag.*, June 1998, pp. 112-119.

[14] ISO/IEC FDIS 14496-2, *MPEG-4 Visual*, Jan. 1999.

[15] S.M. Park et al, "A Low Power Mixed Mode Motion Estimation for Data Compression," *IEEK 2001 SOC Design Conf.*, vol. 1, Nov. 2001, pp. 23-26.

[16] ARM Processor family, http://junitec.ist.utl.pt/chipdir/fam/arm/.

[17] ARM, *Writing Efficient C for ARM*, Doc. Num.: ARM DAI 0034A, 1998.

[18] Hynix Semiconductor Inc., *HB7121B CMOS Image Sensor Data Sheet*, 2001.

[19] ARM, *AMBA Compliance Test Suite*, Doc. Num.: ARM DUI 0006A, 1998.

[20] T. Hopes, "Hardware/Software Co-verification, an IP Vendors Viewpoint," *Proc. Int'l Conf. Computer Design: VLSI Computers and Processors, ICCD*'98, Oct. 1998, pp. 242-246.

[21] S.M. Park et al., "A Single-Chip Video/Audio Codec for Low Bit Rate Application," *ETRI J.*, vol. 22, no. 1, Mar. 2000, pp. 20-29.

[22] Yamamoto et al., "A Scalable MPEG-4 Video Codec Architecture For IMT-2000 Multimedia Applications," *ISCAS2000*, vol. 2, May 2000, pp. 188-191.

[23] Takahashi et al., "A 60-MHz 240-MW MPEG-4 Videophone LSI with 16-Mb Embedded DRAM," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, Nov. 2000, pp. 1713-1721.

[24] Matsushita, http://www.matsushita.co.jp/corp/news/official.data/ata.dir/ en010206-2/en010206-2.html.

[25] Emblaze, http://www.zapex.co.il/z4520.shtml.

Seong-Min Kim received the BS degree in electronic engineering from Kyungpook National University in 1982. In 1983 he joined the Semiconductor Division, KIET, which was consolidated to ETRI in 1985. During 1983 through 1995 he was engaged in semiconductor quality and reliability research. From 1997 he began developing motion estimation and compensation and quantization functional blocks for H.263 video codec, and from 1999 to 2001, developed fine motion estimation and compensation blocks for MPEG-4 video codec. Currently, he is responsible for the time deinterleaver block hardware implementation for digital audio broadcasting for the High Speed Telecommunication IC Design Team.

Ju-Hyun Park received his BE, ME, and PhD from Chonnam National University in 1993, 1995 and 1999. From 1999 to 2002, he was with ETRI in Daejeon, Korea, where he worked as a research scientist in the VLSI Architecture Team. Since 2002, he has been engaged with Mamurian Design Inc**.** (http://www.mamurian.com), in Seoul, Korea and has been involved in the research and development of MPEG-4/JPEG multimedia silicon chips. His research interests include digital video compression and video coding for telecommunication.

Seong-Mo Park received the BS and MS degrees in electronics engineering from Kyungpook National University in Daegu, Korea, in 1985 and 1987. From 1987 to 1992, he was with the LG semiconductor company in Gumi, Korea, where he worked on ASIC design and Mask ROM design. In 1992, he joined the Electronics and Telecommunications Research Institute (ETRI) in Daejeon, Korea where he worked on the development of ASIC design. He is working toward the PhD degree in electronics engineering from Kyungpook National University in Korea. He is currently engaged in research on SoC design, image compression algorithms and SoC architecture design. His main research interests are video coding, image compression, and low power SoC architecture design.

Bon-Tae Koo is a Senior Engineer of the Electronics and Telecommunications Research Institute (ETRI). He received a MS from Korea University in 1991. From 1991, he spent 7 years at System IC Laboratories of the Hyundai Electronics Company, where he researched IC design for communications and multimedia systems. From 1997 to 1999, he was an IC Design Engineer at Dongbu Electronics Company. He joined ETRI in 1999 where he is interested in SoC design for wireless multimedia applications.

Kyoung-Seon Shin received the BS and MS degrees in electronics engineering from Chonbuk National University, Jeonju, Korea, in 1989 and 1991. From 1991 to 1999, he was with the LG semiconductor company in Cheongju, Korea, where he worked on MCU design and application. In 1999, he joined Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea, where he currently works in SoC design methodology development and wireless multimedia SoC design.

Ki-Bum Suh received his BS, MS, PhD degrees in physics from Hanyang University in Seoul, Korea in 1989, 1991, and 2000. In 2000, he joined the Electronics and Telecommunications Research Institute (ETRI) in Daejeon, Korea. He was engaged in the development of MPEG-4 ASIC design, image compression algorithms and VLSI architecture for video codec. He is currently in the Department of Electronics at Woosong University in Daejeon, Korea.

Ig-Kyun Kim received the BS and MS degrees from Kyungbook National University in Daegu, Korea, in 1978 and 1980. Since joining the Electronics and Telecommunications Research Institute (ETRI) in Daejeon, Korea, in 1984, he has been engaged in the research and development in the field of VLSIs. His recent research involves the development of processor systems for video signal processing including high-speed signal processor architecture.

Nak-Woong Eum received the BS degree in electronic engineering from Kyungbook National University in Daegu, Korea, in 1984 and the MS and PhD degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in Daejeon, Korea, in 1987 and 2001. He joined ETRI in Daejeon, Korea, in 1987, in the area of electronic design automation responsible for developing physical synthesis tools. His current research interests include new design methodologies and tools for high-performance communication ICs. He is a member of IEEE.

Kyung-Soo Kim received the BS degree in electronics engineering from Sogang University in Seoul, Korea in 1977. He joined ETRI in 1977. He worked mainly on chip testing, process development and VLSI design. He is currently the Director of Intergrated Circuits Department in Semiconductor Division of ETRI. His research interests include CAD software and ASIC design for wireless communication and ATM systems.